Project Management Mastery with Xcode

Making your project work for you

Session 408

These are confidential sessions—please refrain from filming or taking pictures.

Apple Applications

Project Management Mastery with Xcode

Making your project work for you

Anders Bertelrud
Xcode Architect

http://developer.apple.com/wwdc2005/

Introduction

- Simple projects are straightforward to set up
- Xcode provides great flexibility for complex projects

What Is a “Complex Project”?

- Many targets of different kinds
- Cross-project references
- Embedded frameworks and plug-ins
- Custom build configurations
- Shell script build phases and external targets
- Other special build requirements

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

What We Won’t Cover

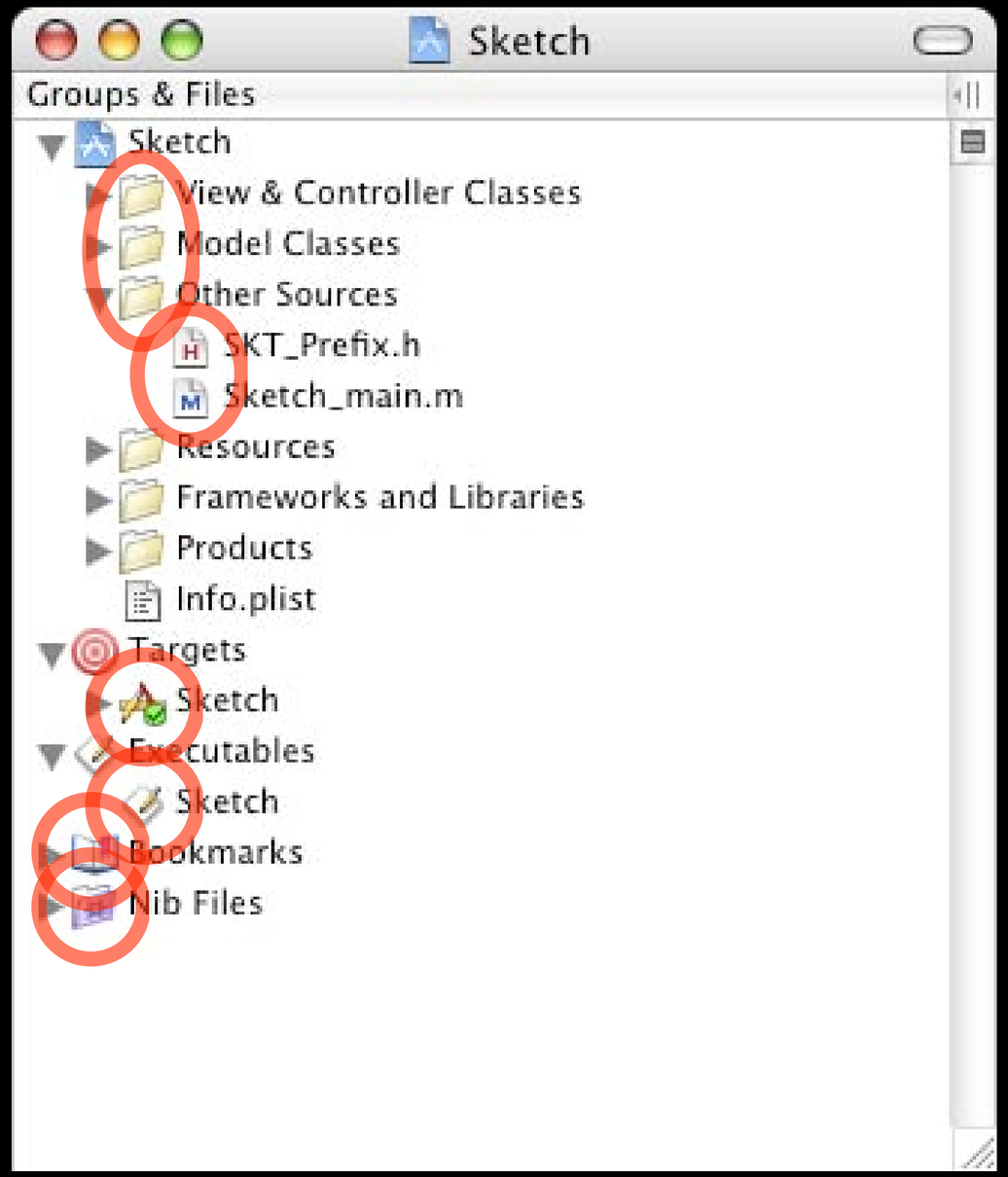
- File editing
- File navigation
- Object modeling
- Project navigation
- Class browsing
- Debugging

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

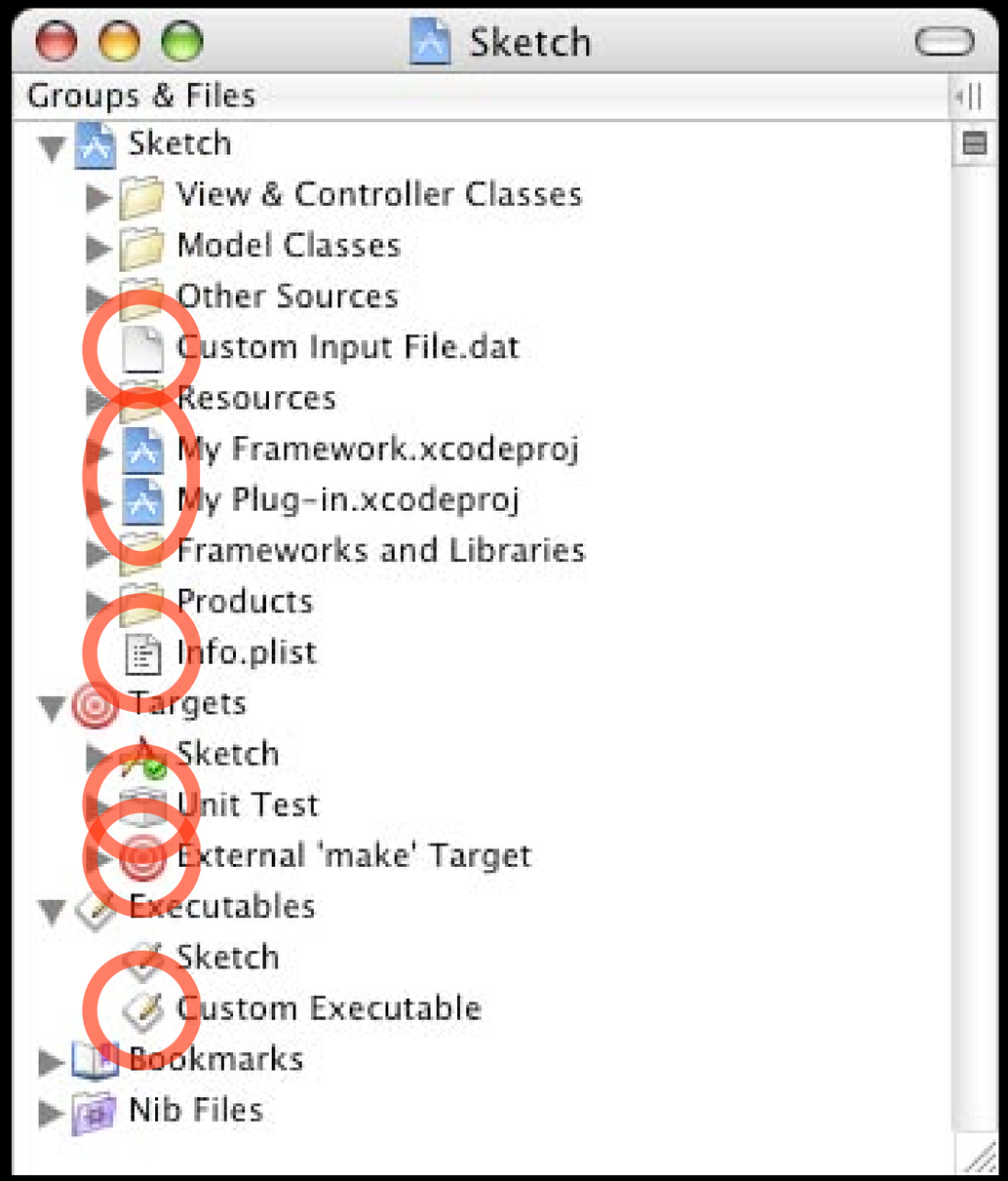
What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets



What's in a Project?

- Almost all projects have



What's in a Project?

- Complex projects might also have

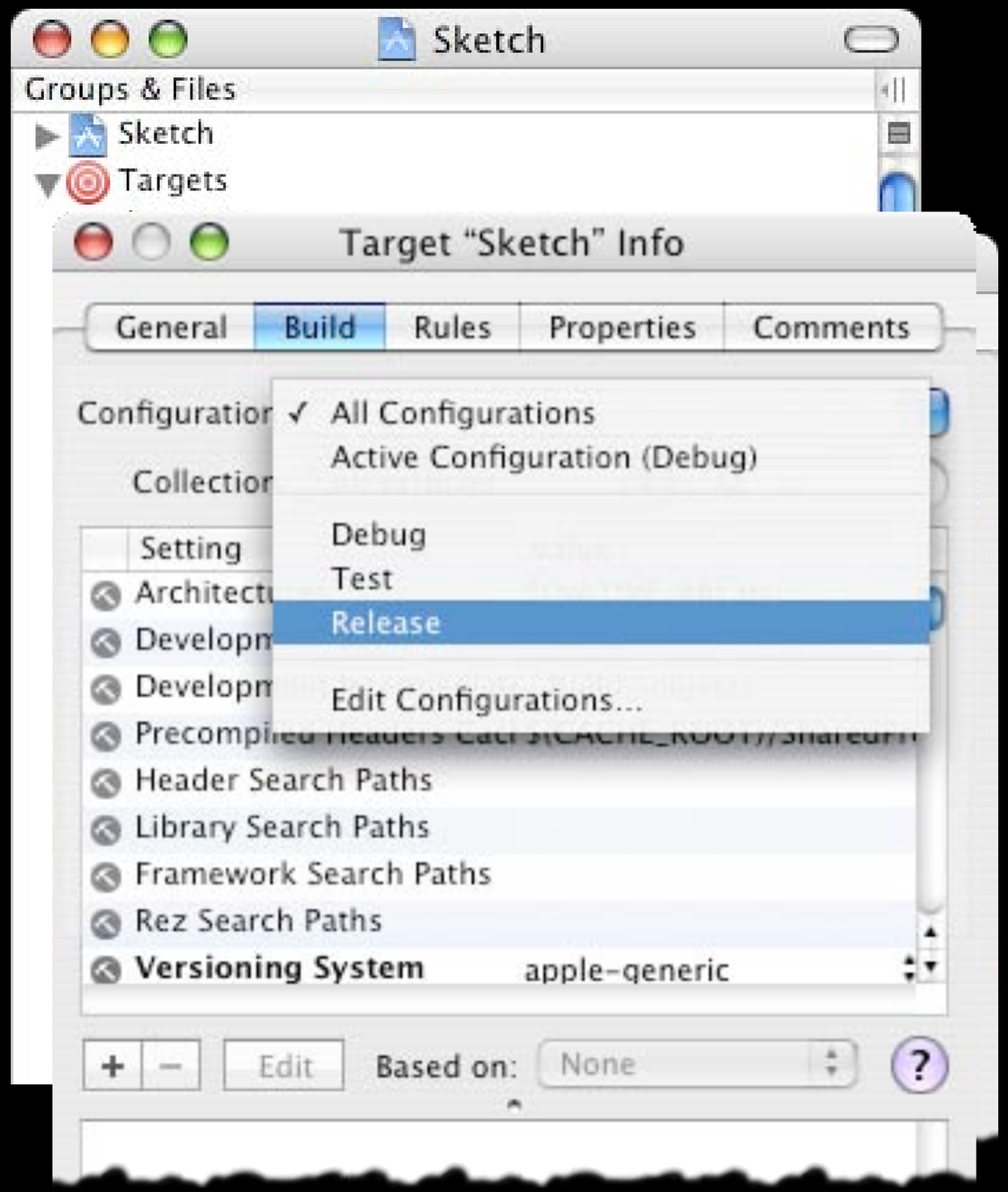
Project References

Advantages

- Divide large projects into small, manageable pieces
- Localize project configuration
- Targets can depend on targets in other projects
- Build all targets in all projects with a single action
- Reduce risk of SCM merge conflicts

Targets

- Main unit of organization for the build system
- Each target defines one logical product
  - e.g. application, framework, plug-in
- A target is like a recipe
  - Its “ingredients” are the files it refers to
  - Its “instructions” are its build phases and rules
- Targets can depend on other targets

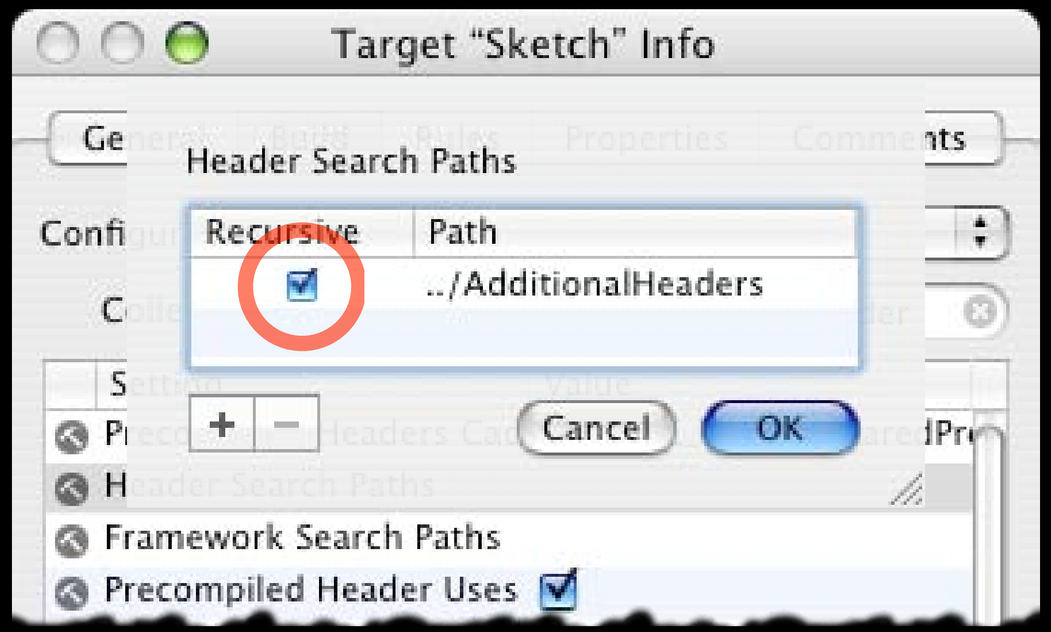


What's in a Target?

- Ingredients

Header Files

- Should I add my headers to the target?
  - Only add headers that need to be embedded in the product
  - No need to add headers just so that source code can find them
- All headers in the project are automatically accessible
- Xcode 2.0 supports recursive search paths



Build System

- Xcode has a fully integrated “native” build system
- Project and source code changes are reflected instantly
- Build system even starts compiling while you’re still typing

Build Phases

- Standard types
  - Copy Headers
  - Copy Resources
  - Compile Sources
  - Link
- Custom types
  - Copy Files
  - Run Script

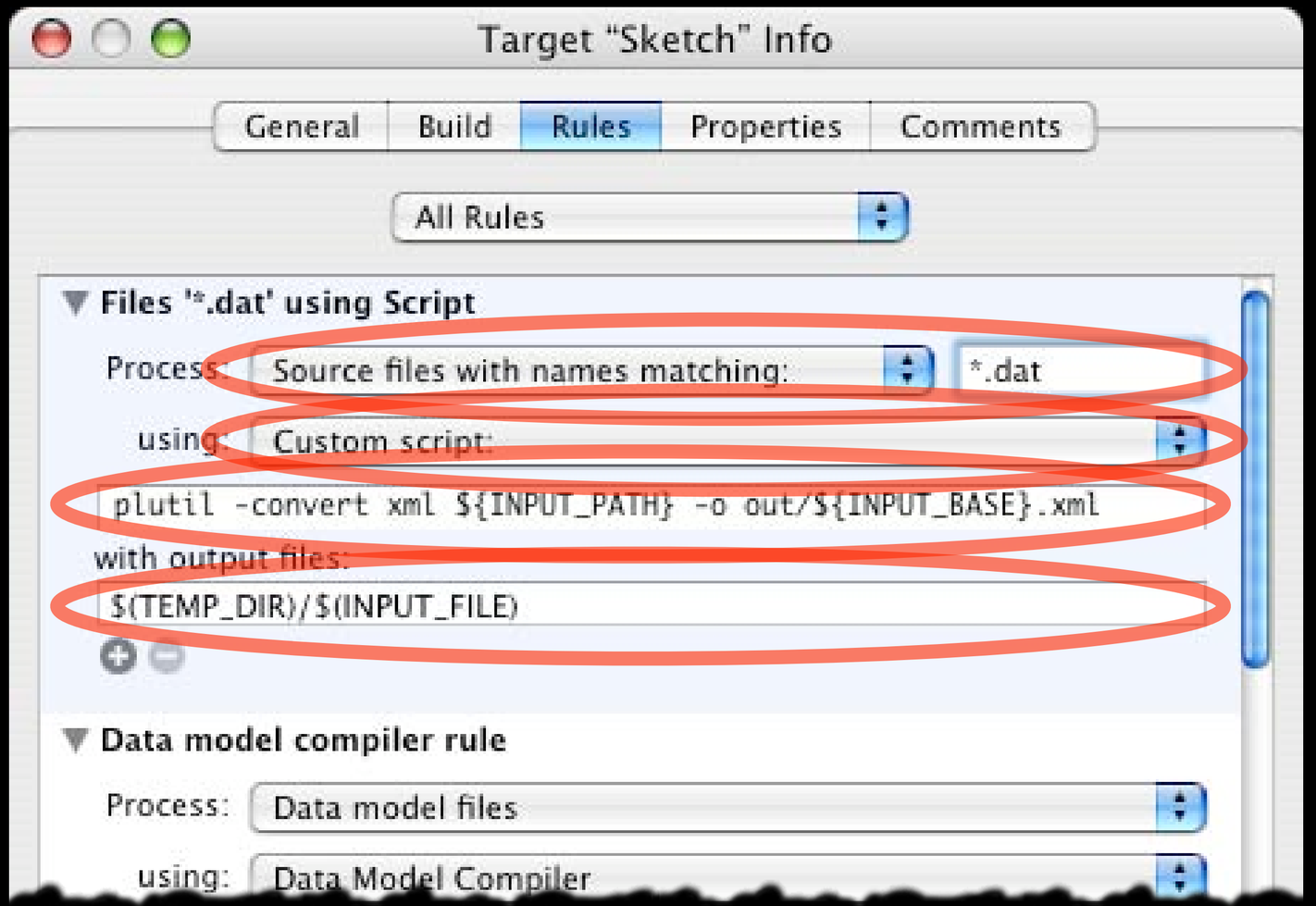
Build Rules

- Build rules specify how source files are processed
- Every rule has a condition and an action
- For each file, the first rule with a matching condition is used
- Use build rules to select compiler version
  - e.g. GCC 3.3 vs. GCC 4.0
- Also useful for generating source code from custom input files

Build Rule Condition and Action

- Condition selects files on which to operate
  - File type
    - e.g. “All Nib files”
  - File name pattern
    - e.g. `*.dat`, `Debug*.c`
- Action specifies what to do with those files
  - Invoke compiler
    - e.g. GCC 4.0 Compiler
  - Run custom script
    - e.g. `plutil -convert xml ${INPUT_PATH} -o out/${INPUT_BASE}.xml`

Build Rule Example



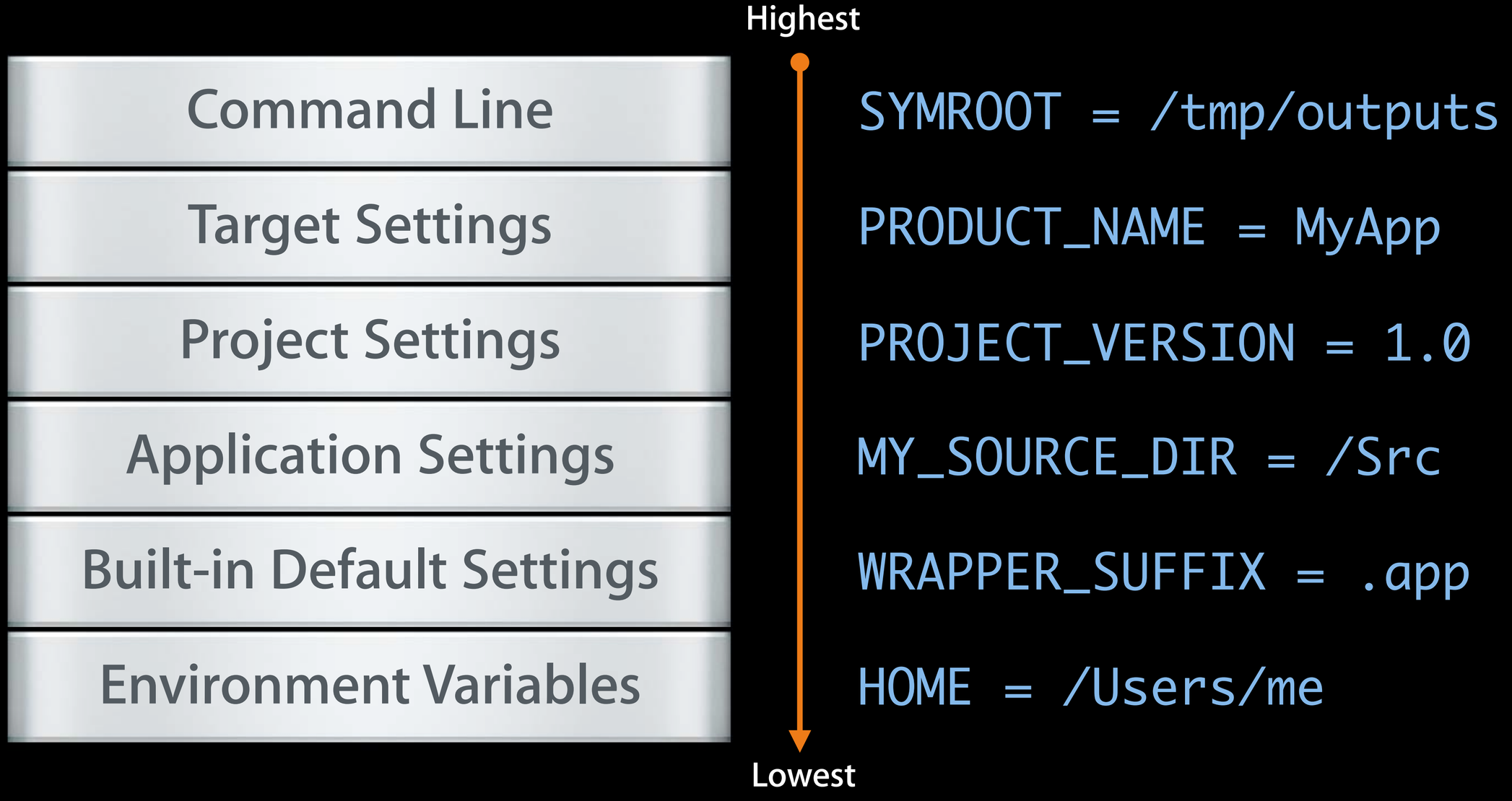
Useful Settings in Build Rules

- Input Files
  - `INPUT_FILE_PATH`—e.g. /src/MyProject/SourceFile.m
- Variants
  - `INPUT_FILE_DIR`—e.g. /src/MyProject
  - `INPUT_FILE_NAME`—e.g. SourceFile.m
  - `INPUT_FILE_BASE`—e.g. SourceFile
  - `INPUT_FILE_SUFFIX`—e.g. .m
- Build Locations
  - `TARGET_BUILD_DIR`—folder in which current target's product is produced
  - `BUILT_PRODUCTS_DIR`—folder containing symlinks to all built products

Build Settings

- Build settings are "knobs and switches"
- Most of the time, static values are appropriate
- A setting's value can also refer to other settings
  - e.g. `PRODUCT_NAME = MyApp_v$(VERSION_NUMBER)`
- Passed as environment variables to tools and scripts
- Can be set at either project or target level levels
- Precedence ordering of build settings
  - Higher levels override lower levels

Precedence of Build Settings



Build Setting Macro Expansion

- References to other build settings use `$(X)` notation
- Expansion occurs in the context in which the value is used
  - Same macro reference can expand differently in different contexts
  - e.g. `$(INPUT_FILE_PATH)` is different for every source file
- Special case: `$(value)` expands to the overridden value
- References to undefined settings are removed

Build Configurations

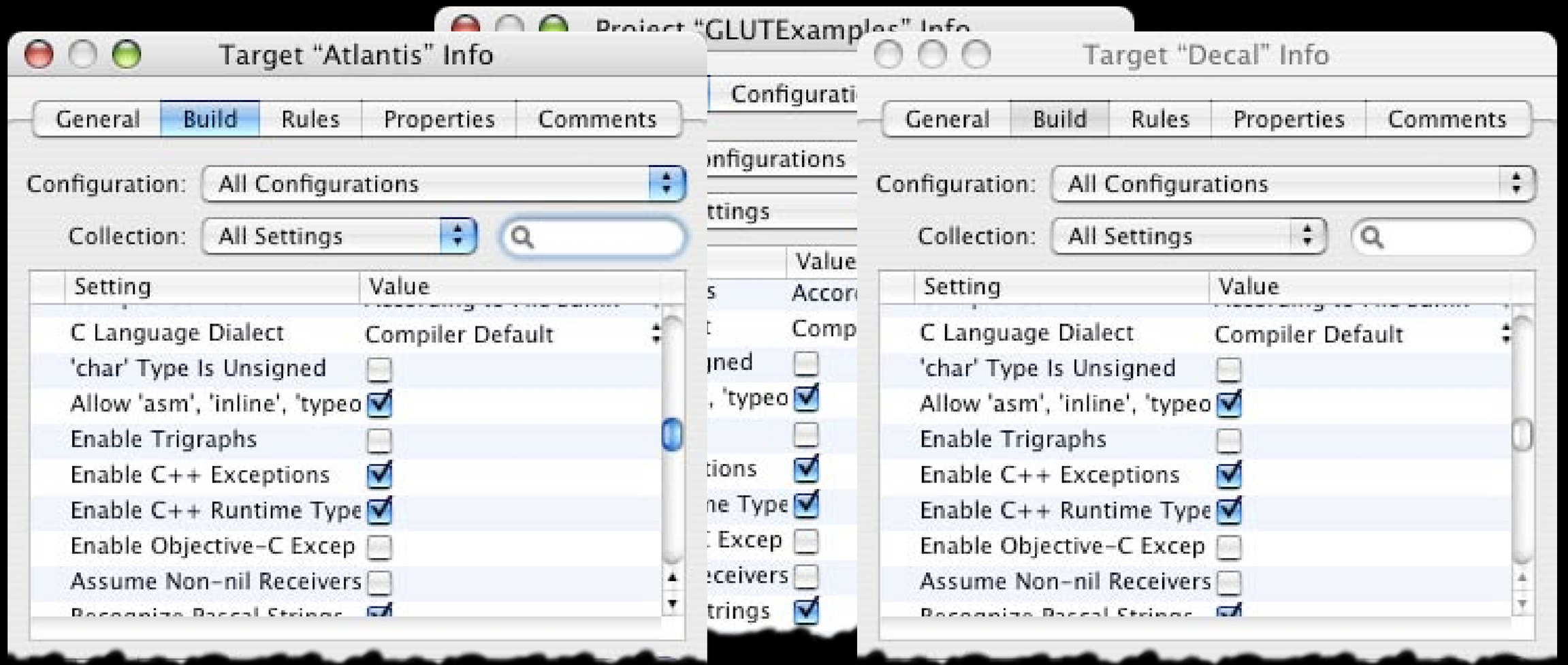
- Build settings are defined in named *configurations*
- Each build configuration has
  - Name (unique within a project)
  - Build settings
- New projects have configurations named “Debug” and “Release”
- Upgraded projects also have “Default” configuration
- Can have as many configurations as you like with any names you want

Build Configurations (Cont.)

- The project and each target has the same set of build setting names
  - Analogy: Method interface
- Each individual build configuration defines its own meaning
  - Analogy: Method implementation

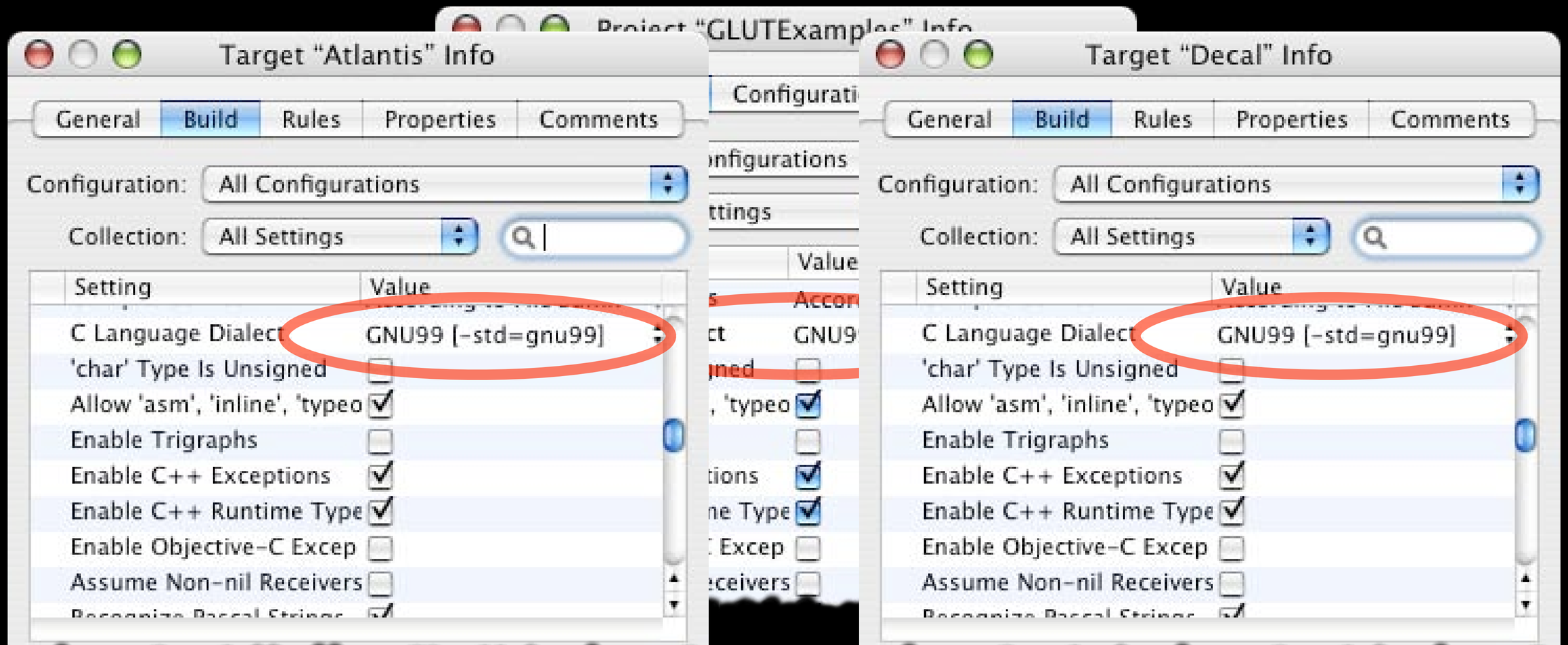
Settings at the Project Level

- All targets in the project inherit build settings defined at the project level



Settings at the Project Level

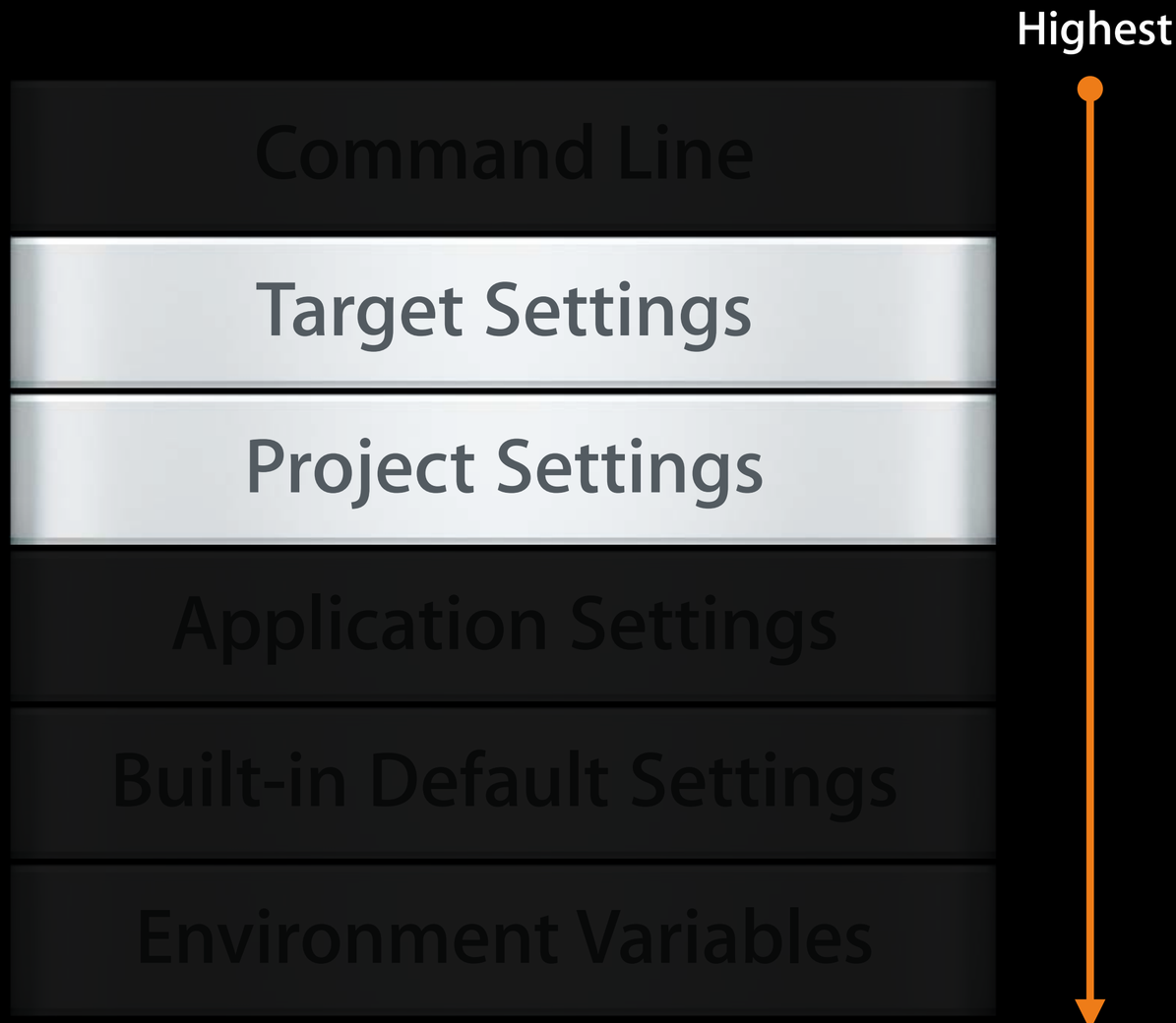
- All targets in the project inherit build settings defined at the project level



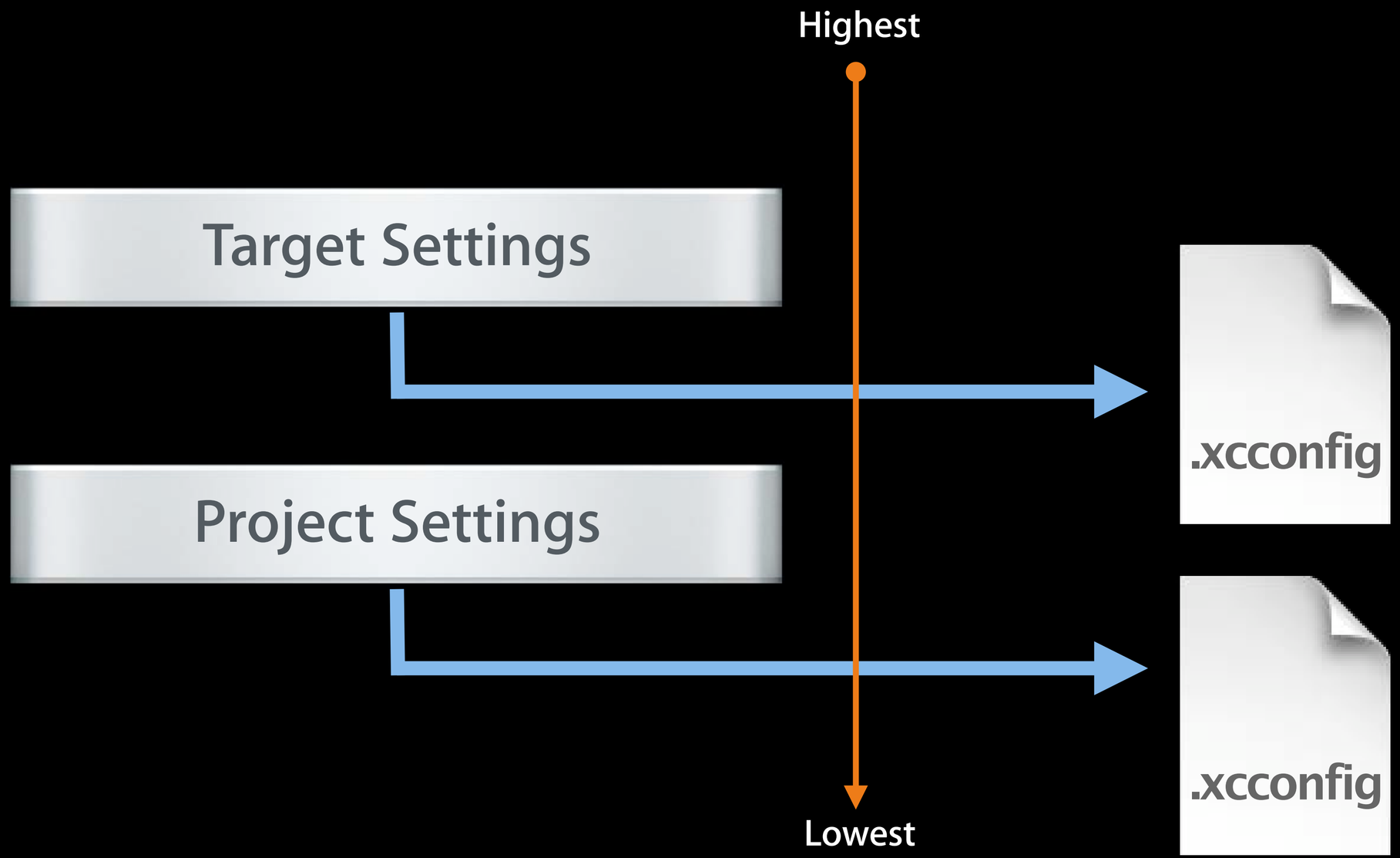
Basing Build Configurations on .xcconfig Files

- Include .xcconfig text files in your project
- Contains a list of `SETTING=VALUE` pairs
- Prototype with drag-and-drop from inspectors
- Multiple targets can share the same config file
- Reuse the .xcconfig files across projects

Build Setting Precedence Revisited

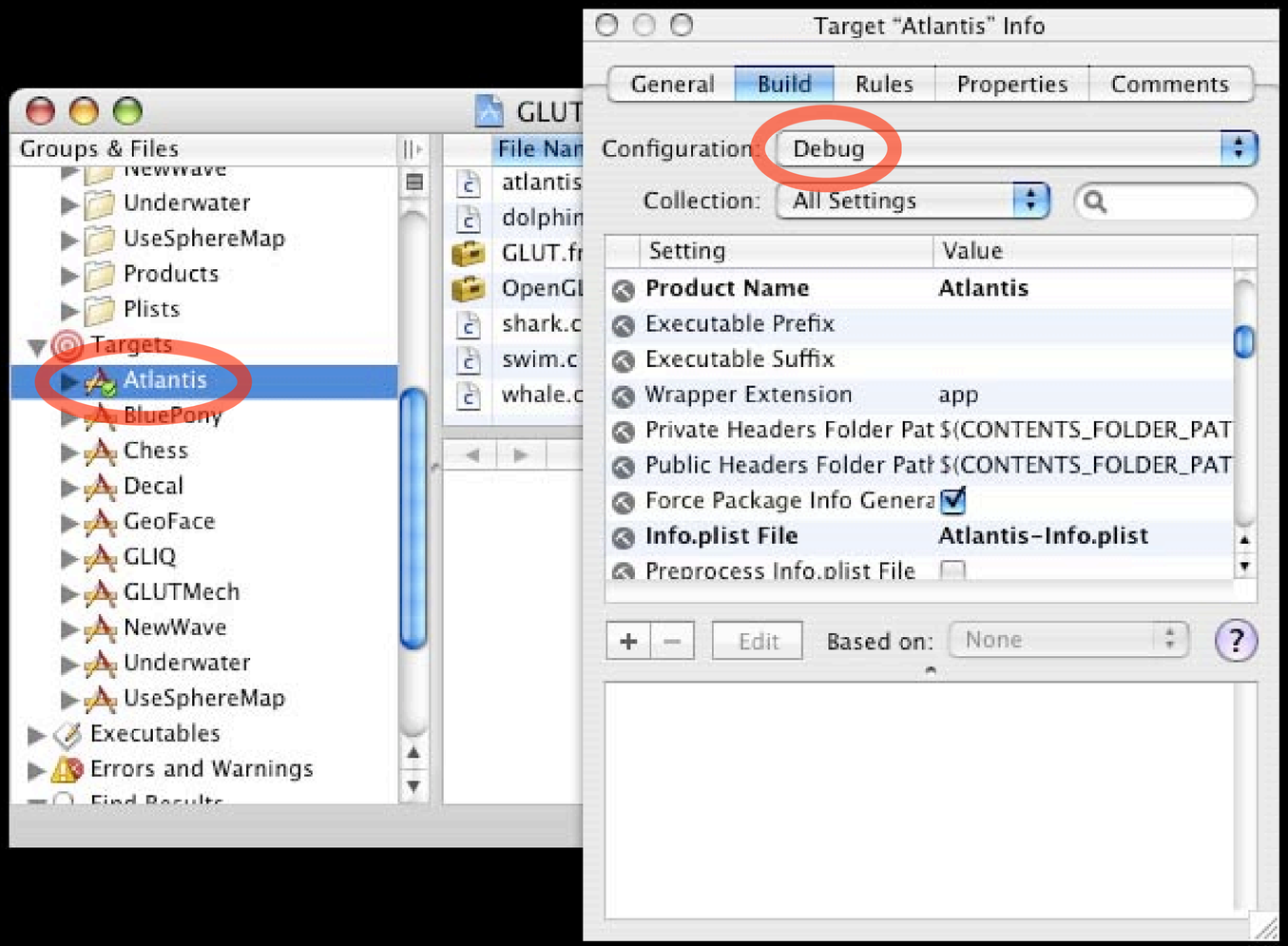


Build Setting Precedence Revisited



Effective Use of the Settings Inspector

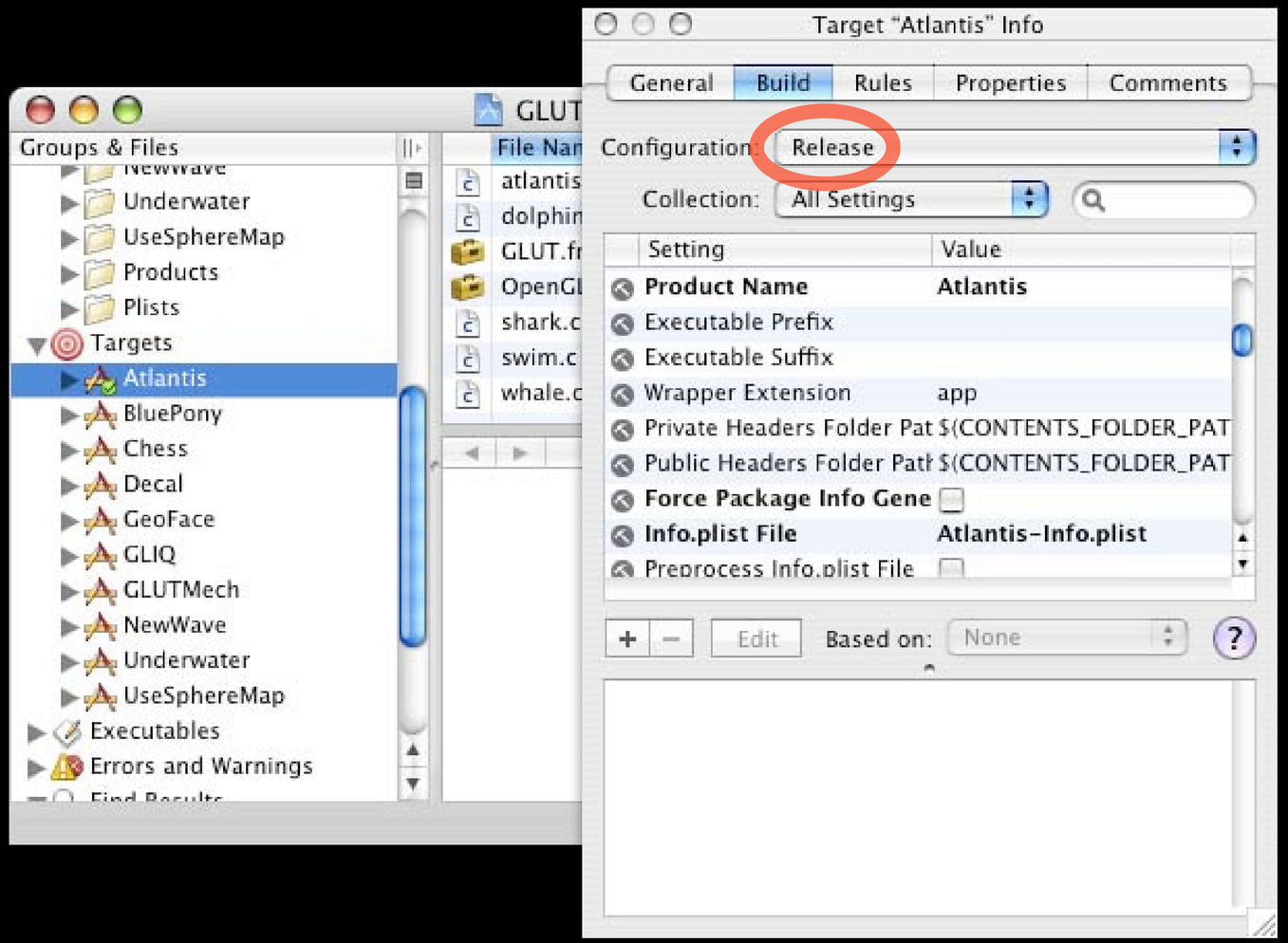
Making sure the settings end up where you want them



	Debug	Test	Release
Atlantis			
BluePony			
Chess			
Decal			
GeoFace			
GLIQ			
GLUTMech			
NewWave			
Underwater			
UseSphereMap			

Effective Use of the Settings Inspector

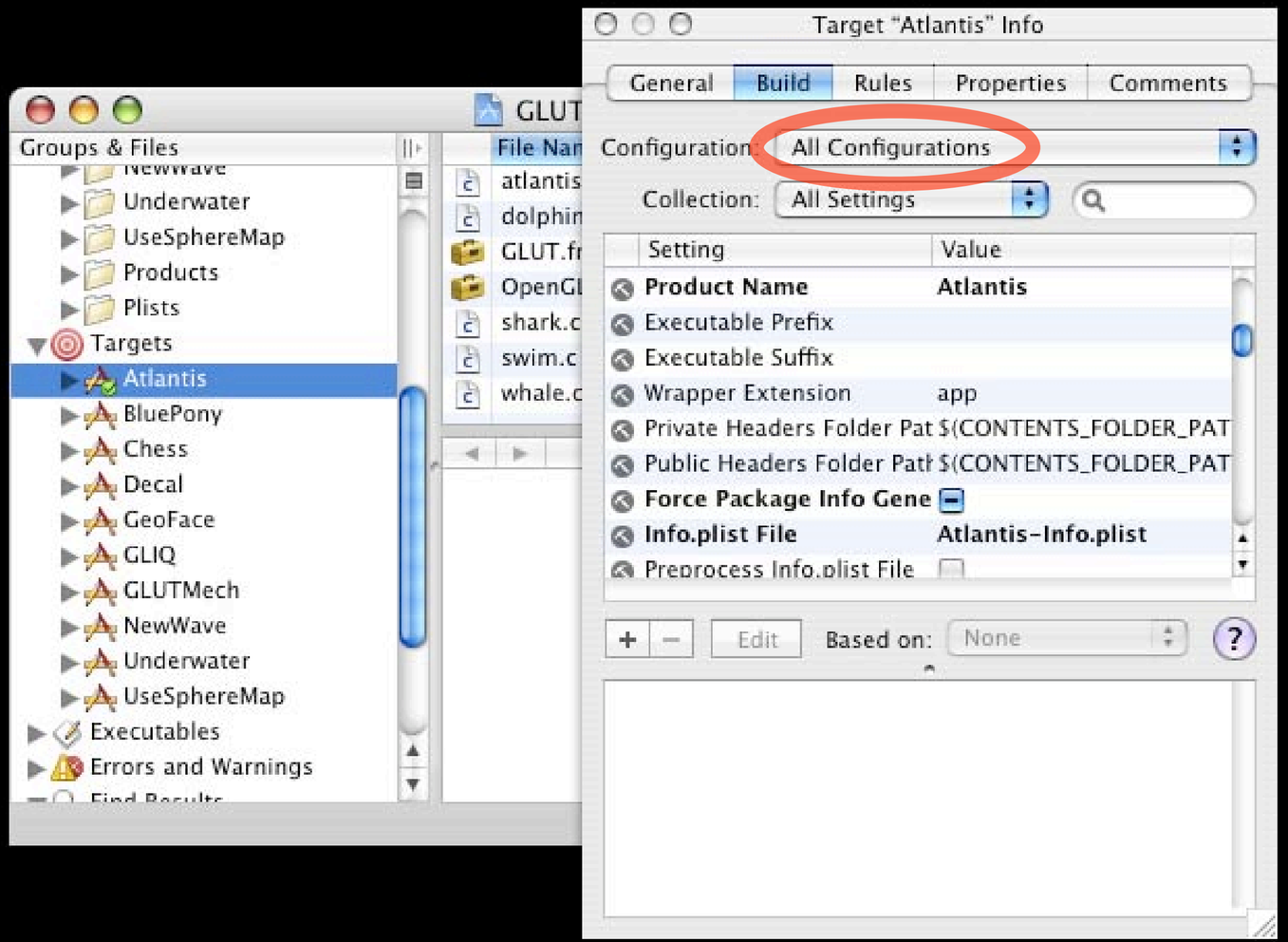
Making Sure the Settings End Up Where You Want Them



	Debug	Test	Release
Atlantis			
BluePony			
Chess			
Decal			
GeoFace			
GLIQ			
GLUTMech			
NewWave			
Underwater			
UseSphereMap			

Effective Use of the Settings Inspector

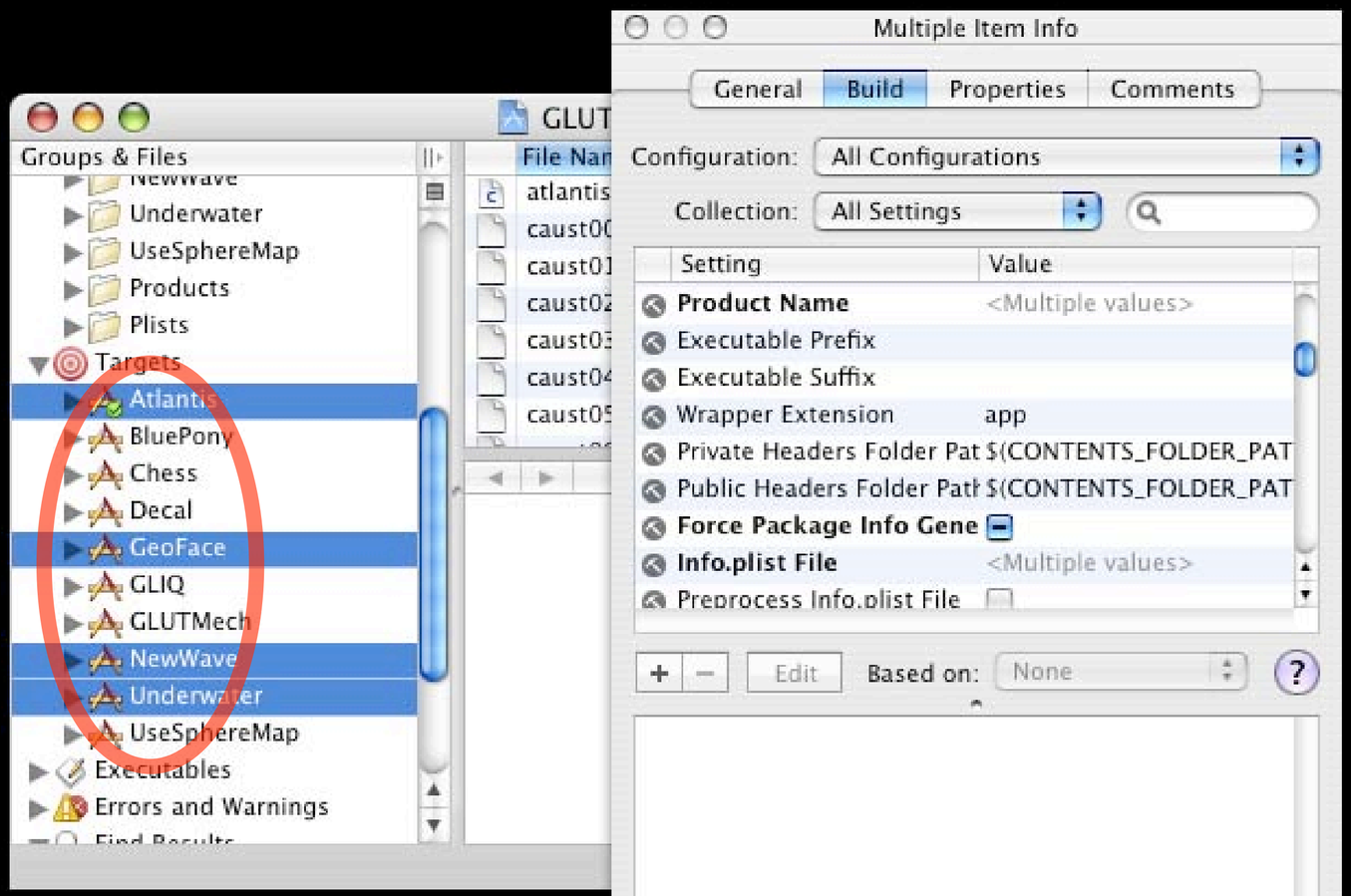
Making Sure the Settings End Up Where You Want Them



	Debug	Test	Release
Atlantis			
BluePony			
Chess			
Decal			
GeoFace			
GLIQ			
GLUTMech			
NewWave			
Underwater			
UseSphereMap			

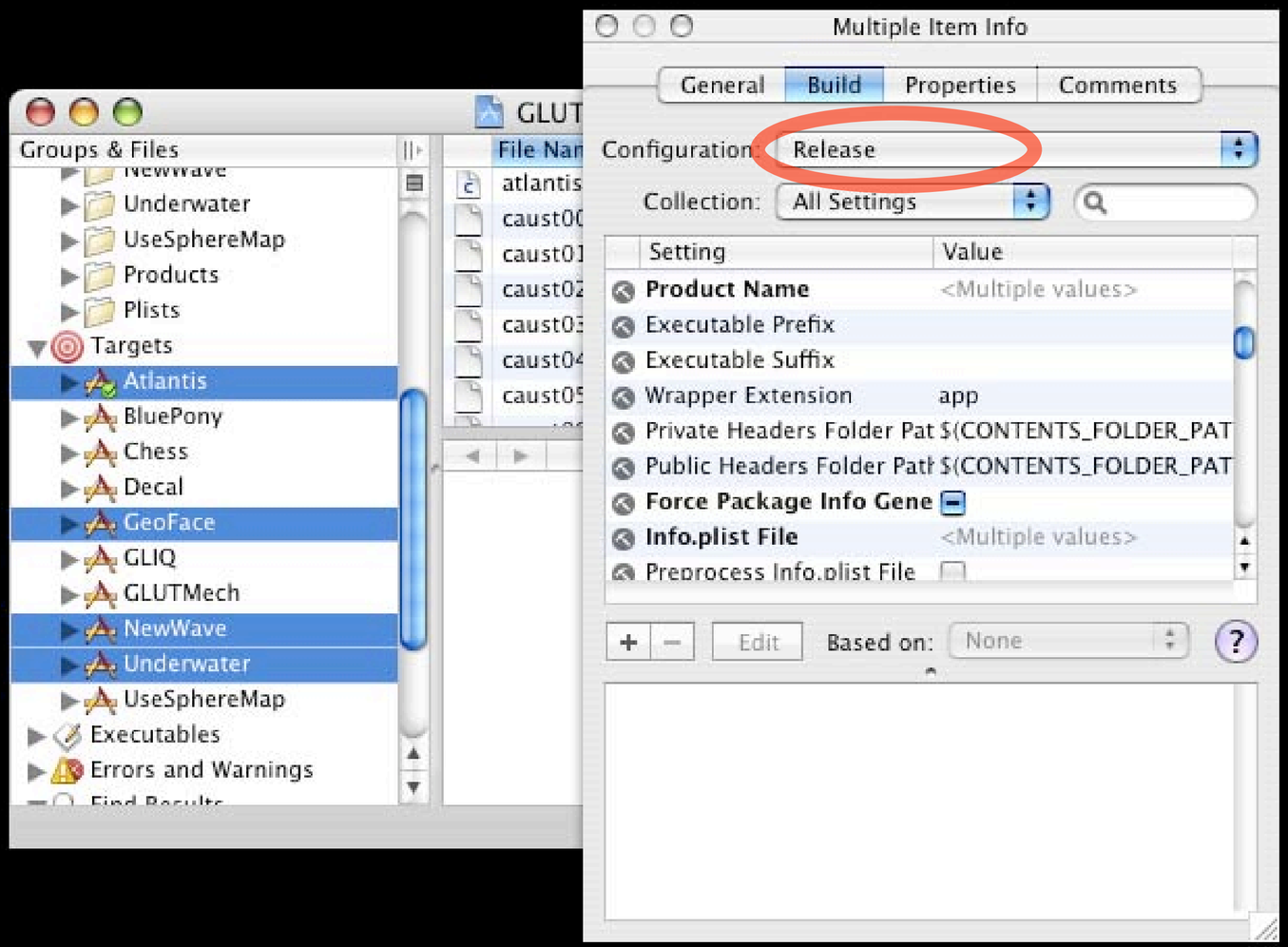
Effective Use of the Settings Inspector

Making Sure the Settings End Up Where You Want Them



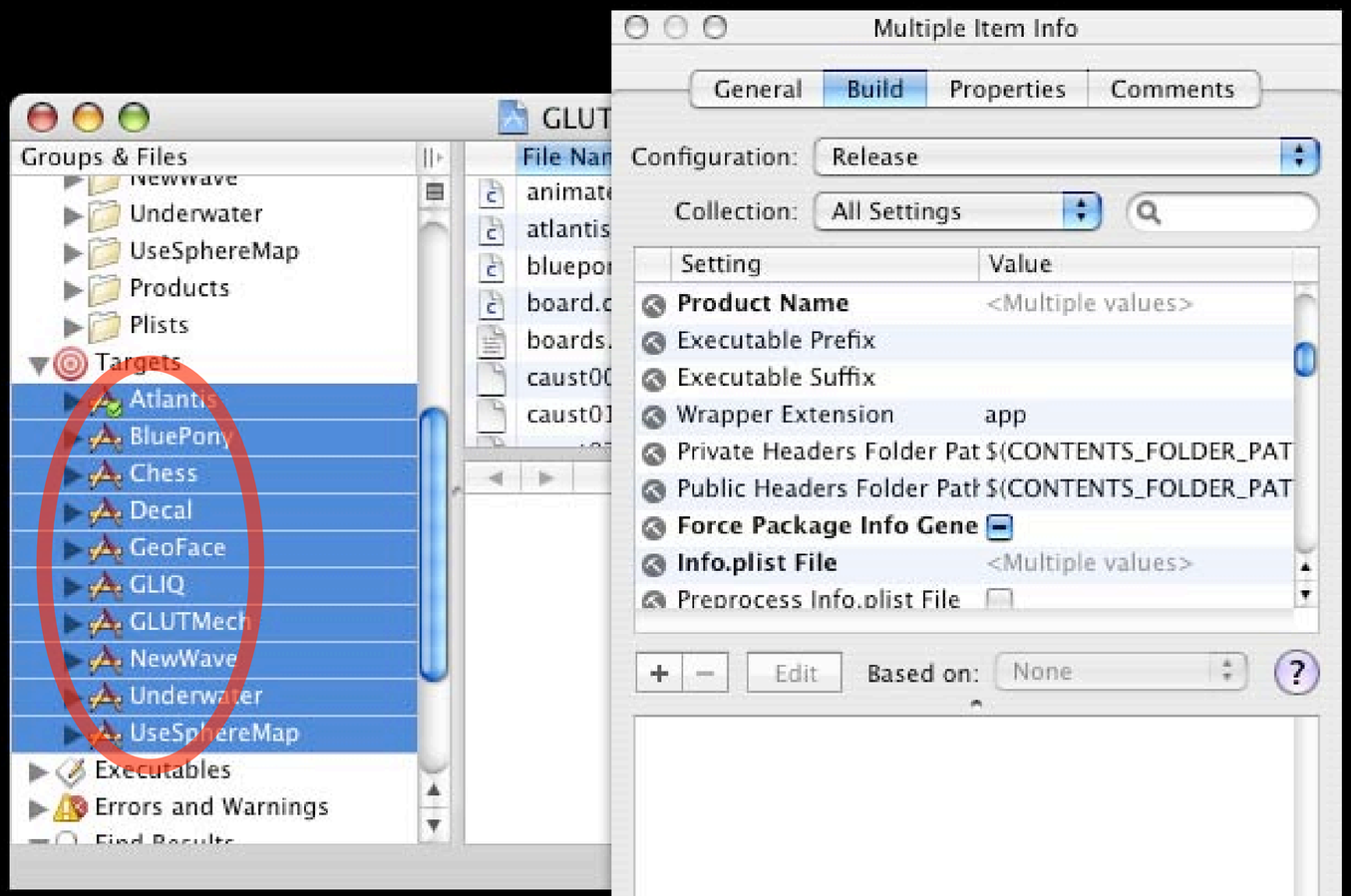
Effective Use of the Settings Inspector

Making Sure the Settings End Up Where You Want Them



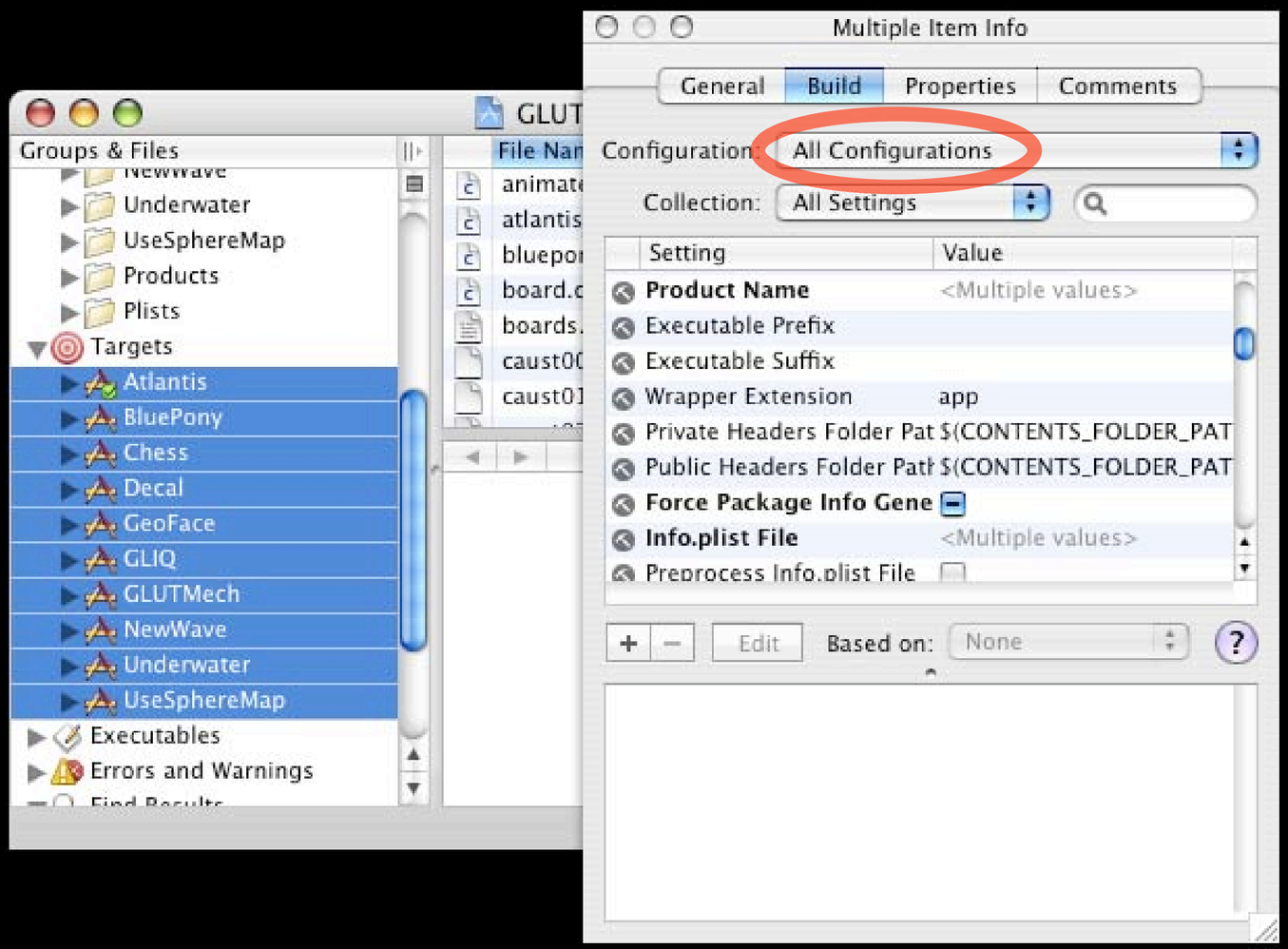
Effective Use of the Settings Inspector

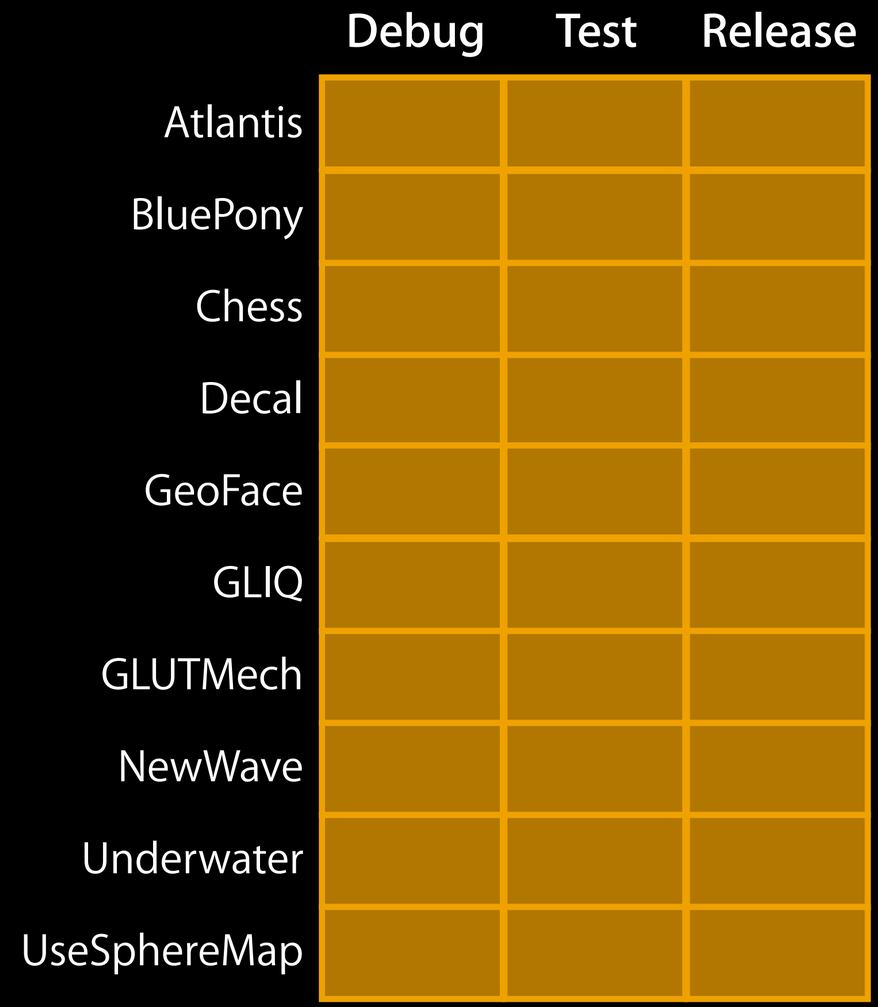
Making Sure the Settings End Up Where You Want Them



Effective Use of the Settings Inspector

Making Sure the Settings End Up Where You Want Them





What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

What's an SDK?

- SDK—"Software Development Kit"
- Provides build-time access to operating system functionality
- Build against a different OS release from what you're running
  - e.g. develop on Mac OS X 10.4 but allow product to run on 10.2.8
- Use of API already implies a range of Mac OS X releases
- Specifying an SDK makes that assumption explicit
  - Chosen SDK defines upper limit
  - "Mac OS X Deployment Target" setting specifies lower limit

Why Should You Use an SDK?

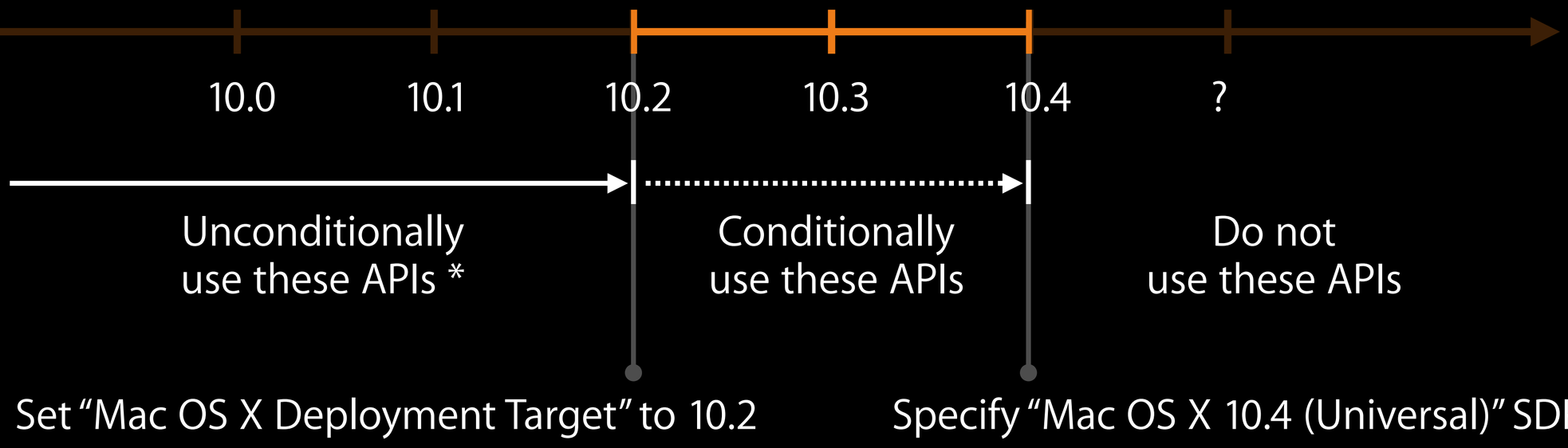
- By default, projects use the system installed on the machine
- Specifying an SDK improves consistency among machines and over time
  - SDKs don’t change over times
- SDKs also lets the build system warn about
  - Use of API that’s “too new”
  - Use of deprecated API

What's in an SDK?

- SDK are sparse directory trees mimicking the system layout
- Headers
  - Same as what shipped with the corresponding Mac OS X release
- Dynamic library stubs
  - Contain no code; only link-time information
- Static libraries
  - Contain code that's statically linked into your software product
- No compilers or other command-line tools

Using SDKs—An Example

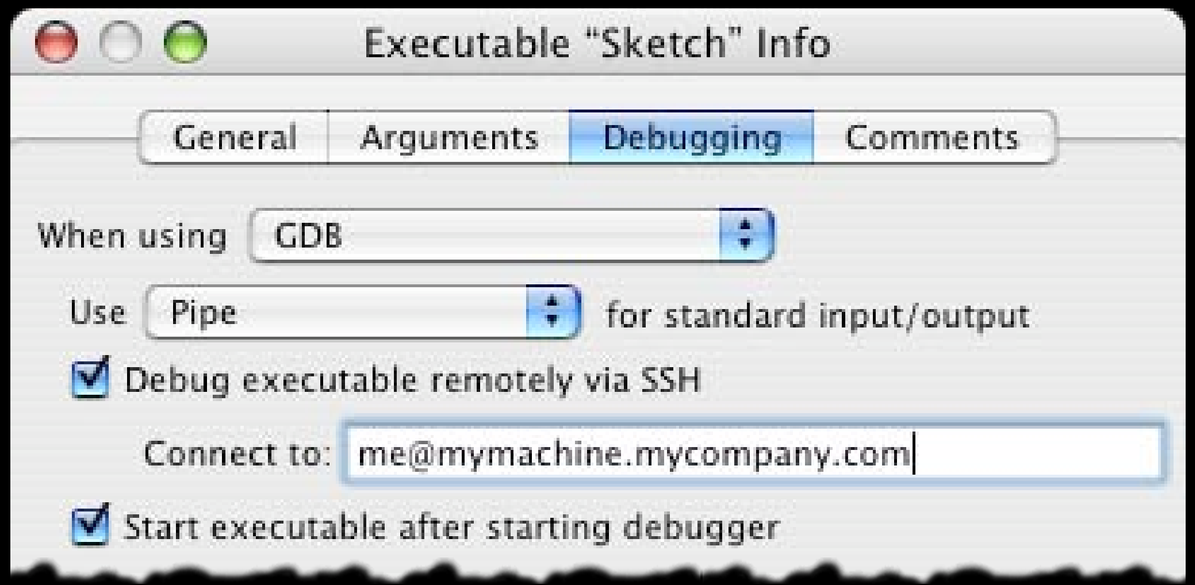
- Consider a project that wants to use 10.4 API but only require 10.2
- Specify 10.4 as the SDK, and set "Mac OS X Deployment Target" to 10.2



* avoid using existing API marked as deprecated

Debugging With SDKs

- Can debug most issues on the system you're running
- Use remote debugging to investigate issues specific to OS version
- Need developer tools installed on the remote machine



What We'll Cover

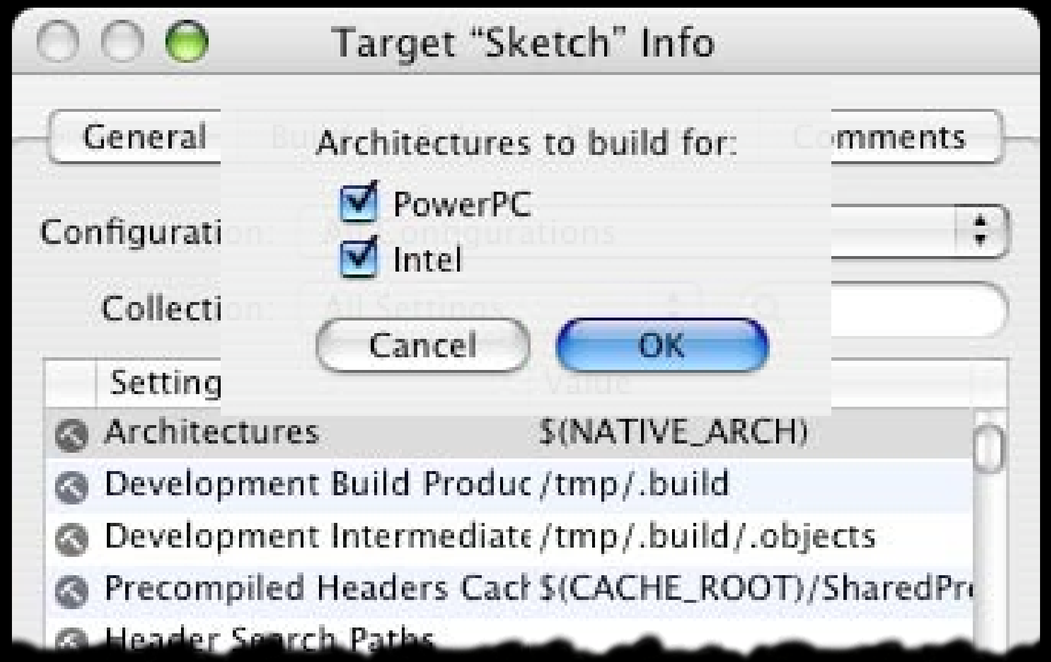
- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

Creating Universal Binaries

- Xcode fully supports development for Intel
- Can create “universal” binaries containing both PowerPC and Intel code
- For many projects, it’s as simple as clicking a checkbox

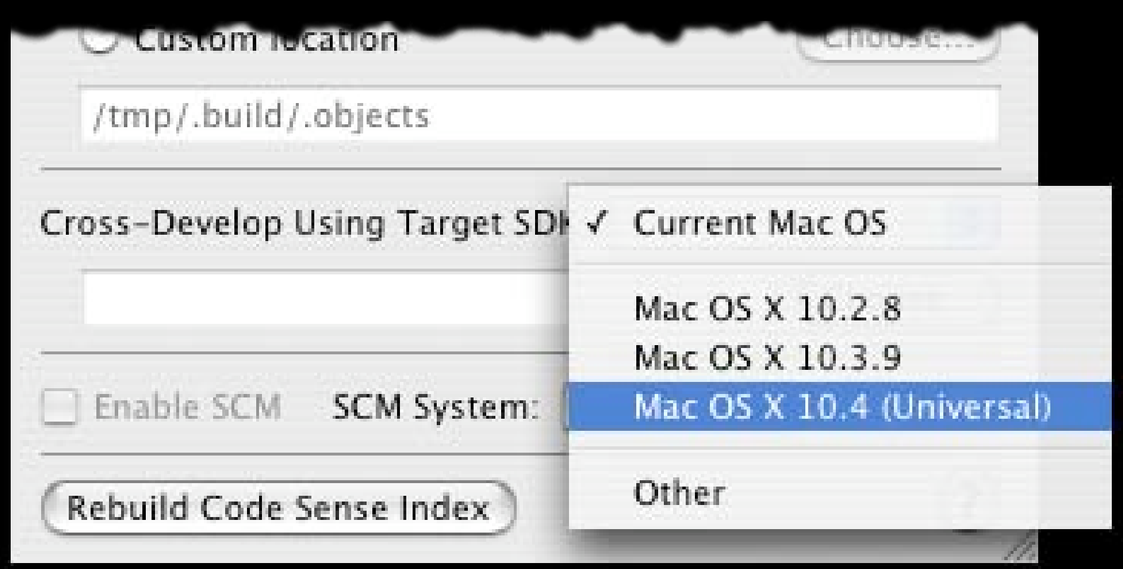


Architecture-Specific Settings

- Some build settings have architecture-specific forms
- Architecture-specific settings can be set at any level
  - Settings at target level affect only the target
  - Settings at project level affect all targets in project
  - *.xcconfig* files can be used to share common settings among projects
- Naming convention
  - Format is `<settingname>_<architecture>`
  - Architecture name is of the same form as in *arch(3)* man page
  - For historical reasons, generic Intel architecture is called “i386”
  - Example: `OTHER_CFLAGS_ppc` vs `OTHER_CFLAGS_i386`

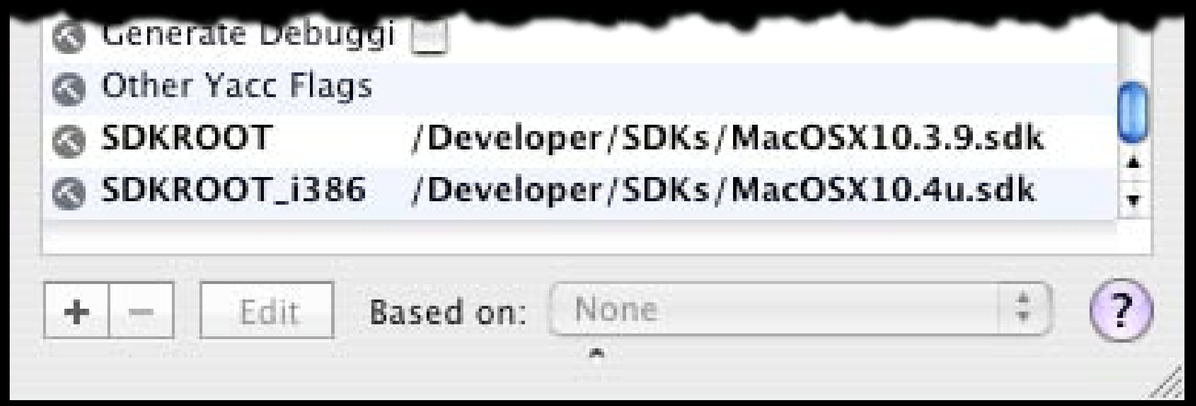
Architecture-Specific SDKs

- Xcode allows a different SDK for each architecture
- Primary SDK
  - Used for indexing and documentation lookup
  - Set in “General” tab of the project inspector, as usual



Architecture-Specific SDKs

- Xcode allows a different SDK for each architecture
- Primary SDK
  - Used for indexing and documentation lookup
  - Set in “General” tab of the project inspector, as usual
- Alternate SDKs
  - Different SDKs can be chosen for different architectures
  - Set in “Build” tab of project or target inspector, depending on desired scope



Settings Configurable Per-Architecture

- Each of these settings can be defined per-architecture
  - SDKROOT_$(arch)
  - OTHER_CFLAGS_$(arch)
  - OTHER_LDFLAGS_$(arch)
  - OTHER_LIBTOOLFLAGS_$(arch)
- If defined, SDKROOT_$(arch) overrides SDKROOT
- Settings containing flags are additive

Ordering of Architecture-Specific Flags

- Architecture-specific flags are emitted after general flags
- For example, if
  - `OTHER_CFLAGS = -Wsequence-point`
  - `OTHER_CFLAGS_i386 = -mno-fancy-math-387`
- Then the compiler command line for Intel builds is

```
... -Wsequence-point -mno-fancy-math-387 ...
```

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

What We'll Cover

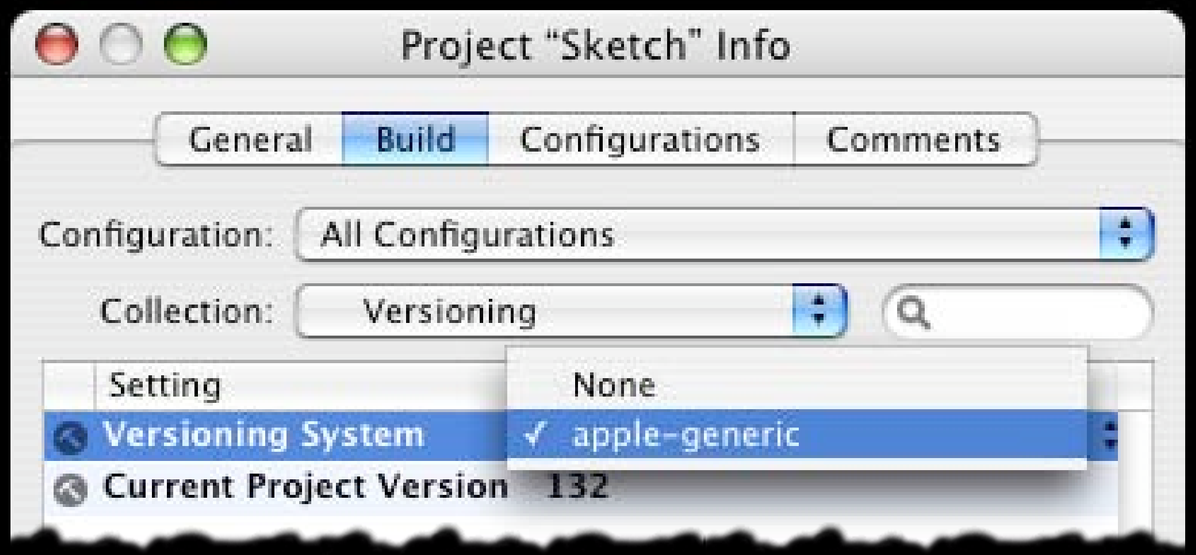
- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

Project Versioning

- Almost all software products have a version number
- It is often useful to also have a build number
- Command line tool *agvtool* helps automate the process
- Tool supports both “marketing” and “build” version numbers

Configuring Project Versioning

- Configure your project to use “Apple Generic” versioning system
- Version number can be anything you want
  - Often useful to have it be the build number



Using agvtool

- Command line tool *agvtool* operates on build settings in project
- Handles both marketing version and build number
- Can automatically check in project changes to CVS
- Best to close project before running agvtool

```
agvtool help
agvtool what-version|vers [-terse]
agvtool next-version|bump [-all]
agvtool new-version [-all] <versNum>
agvtool tag [-force|-F] [-noupdatecheck|-Q]
agvtool what-marketing-version|mvers [-terse]
agvtool new-marketing-version <versString>
```

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

Fine-Tuning Your Targets

- Precompiled prefix headers (PCHs)
- *Info.plist* processing

Precompiled Headers

- Usually, most code seen by the compiler comes from headers
- A large set of unchanging headers is often included by all source files
  - e.g. every header reachable from Carbon.h or Cocoa.h
- Example: UCarbonEvent.cpp under /Developer/Examples/
  - 54 lines of code in UCarbonEvent.cpp itself
  - 90944 total lines of code seen by the compiler after preprocessing
  - Only ~0.06% “interesting” content!
- Solution: precompile the stable headers

Setting Up Precompiled Prefix Headers

- Set at either the project or the target level
  - Create a prefix header that includes common API
  - Enable the “precompile prefix header” setting
- Xcode precompiles the prefix header on-demand
- Precompiled header is automatically shared between projects when possible
- Keep in mind
  - Prefix header gets included for all source files in target
  - Only put infrequently changed headers in the PCH

Info.plist Processing

- *Info.plist* is a property list file that tells the OS about your product
  - Identifier and icon of bundle
  - Name of Mach-O executable
  - Types and icons of documents
  - Version numbers
- Some entries are localized in an *InfoPlist.strings* file
- Some of the information is already in build settings at build time
- Xcode lets you
  - Refer to build settings from within the Info.plist file
  - Run the Info.plist file through the GCC preprocessor

Info.plist Processing

Build Setting Expansion Example

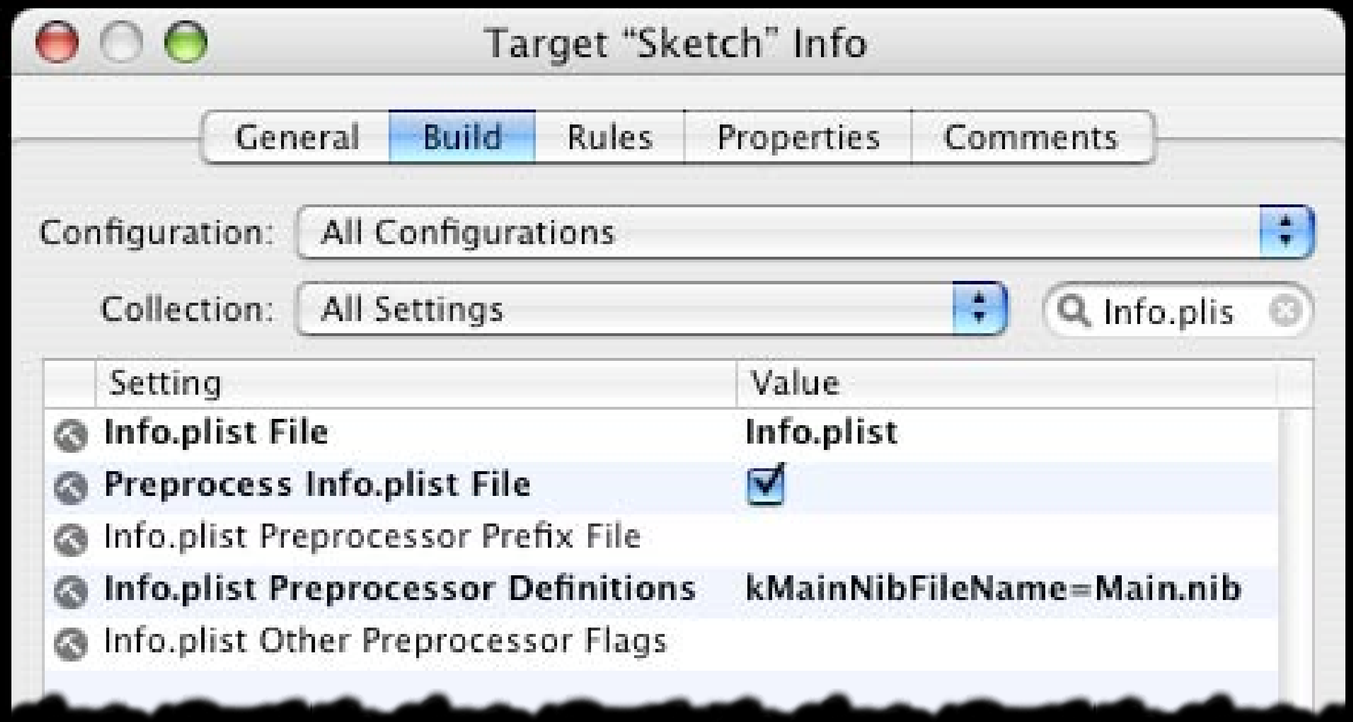
- Happens automatically—no switch needs to be enabled

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>CFBundleExecutable</key>
   <string>$(EXECUTABLE_NAME)</string>
   <key>CFBundleVersion</key>
   <string>$(CURRENT_PROJECT_VERSION)</string>
   <key>CFBundleDocumentTypes</key>
   <array>
      <dict>
         <key>CFBundleTypeExtensions</key>
```

Info.plist Processing

GCC Preprocessing Example

- Enabled using the project or target inspector
- Can also define Info.plist-specific preprocessor macros here



Info.plist Processing

GCC Preprocessing Example

- Use preprocessor directives and preprocessor macro names in Info.plist

```
#include "CommonInfoPlistSettings.h"
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>kMainNibFileName</key>
   <string>Draw2</string>
#ifdef ENABLE_SCRIPABILITY
   <key>NSAppleScriptEnabled</key>
   <string>YES</string>
#endif
   <key>NSPrincipalClass</key>
```

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

Demo

Project Configuration

http://developer.apple.com/wwdc2005/

What We'll Cover

- Project model concepts
- Targeting multiple versions of Mac OS X
- Creating universal binaries
- Keeping track of version numbers
- Fine-tuning your targets

More Information

http://developer.apple.com/wwdc2005/

Documentation, Sample Code, and Other Resources

http://developer.apple.com/wwdc2005/

Related Sessions

Development Tools Feedback Forum	Session 410 Friday 5:00PM

Who to Contact

Matthew Formica
Developer Tools and Cocoa Evangelist
mformica@apple.com

xcode-users Discussion List
Community-Based Support for Xcode Users
xcode-users@lists.apple.com

Q&A

Matthew Formica
Developer Tools and Cocoa Evangelist
mformica@apple.com

http://developer.apple.com/wwdc2005/

WWDC2005

Worldwide Developers Conference